Facultad de ingeniería de la UBA

66.20 Organización de Computadoras

Trabajo práctico N°2
Jerarquías de Memoria

Estudio del comportamiento de la memoria caché para algoritmos de multiplicación de matrices
Segundo cuatrimestre de 2020

Integrante	Padrón	Correo electrónico
Calvo, Mateo Iván	98290	macalvo@fi.uba.ar
Jamilis Netanel David	99093	njamilis@fi.uba.ar
Sabao Tomás	99437	tsabao@fi.uba.ar

Fecha de entrega: 01/12/2020
Grupo 10

Índice

1. Introducción

El motivo de este trabajo práctico fue estudiar el comportamiento de las jerarquías de memoria, más específicamente la memoria caché. Para ello, se estudió teóricamente su comportamiento para un algoritmo de multiplicación de matrices cuadradas, y luego se realizaron simulaciones para analizar el comportamiento "real"de la memoria caché.

2. Análisis previo

Se realizó primero un análisis del algoritmo de multiplicación de matrices provisto por la cátedra. En su implementación en el lenguaje de programación C, para una multiplicación de matrices $C = A \times B$, el algoritmo realiza una cantidad total de accesos a memoria dada por:

- N^3 accesos a la matriz C para lectura del cálculo parcial del producto interno.
- N^3 accesos a la matriz A para lectura.
- N^3 accesos a la matriz B para lectura.
- N^3 accesos a la matriz C para escritura del cálculo parcial del producto interno.

lo cual deja una cantidad total de accesos:

$$\text{Accesos} = 4 \times N^3$$

Además, se optó por implementar un algoritmo que optimice los accesos a la memoria de la matriz resultado, ya que dicha matriz puede ser escrita una única vez por producto interno (entre filas de A y columnas de B) calculado. En esta versión de la multiplicación, se accede en memoria a los elementos de la matriz A y la matriz B para realizar el producto interno, y una vez hecho eso se accede a la matriz C para guardar el resultado. Esta nueva implementación tiene una cantidad de accesos a memoria dada por:

$$\text{Accesos} = 2 \times N^3 + N^2$$

La versión provista por la cátedra (ajustada para matrices cuadradas) y la versión optimizada se encuentran adjuntas a esta entrega digital, respectivamente en **src_mmult_naive_c.c** y en **src_mmult_asm.S**

3. Preparación del entorno de ejecución

Para poder ejecutar el programa es necesario tener `valgrind` y `make` instalados en la máquina virtual de MIPS. Se necesita a su vez una carpeta con el nombre de `inputs` que se encuentre en el mismo directorio en el que se encuentra el archivo `benchmark.sh`. En la carpeta se deberán poner los archivos de texto que contienen a las matrices que se quieren analizar usando `cachegrind`.En la entrega se adjunta un script en lenguaje python `crear_matrices.py` que genera una serie de archivos de texto con el formato de matriz utilizado. Ese script fue el usado para realizar este informe y ejecutar las simulaciones.

Comando de simulación del cache

```
root@debian-stretch-mips:/tp2#: bash benchmark.sh
[ruta de la funcion] [clear_cache] [inputs] [logs] [flags
cachegrind instrucciones] [flags cachegrind data]
```

- `ruta de la funcion`: Ruta absoluta de la función a utilizar para anotaciones con cachegrind.
- `clear_cache`: `clear_cache=no` o `clear_cache=yes` según se quiera forzar un *cold start* para la memoria caché antes de realizar la multiplicación.
- `inputs`: carpeta donde se encuentren los archivos de texto que contienen las matrices.
- `flags cachegrind instrucciones`: Mismo flag que será pasado a cachegrind, para instrucciones.
- `flags cachegrind data`: mismo flag que será pasado a cachegrind, para datos.

Una vez ejecutado, el script compilará el código para la función seleccionada, eliminará la directiva `.file` en el archivo ensamblador automáticamente para que sea compatible con `cachegrind`, ejecutará la simulación para archivo de matrices, y creará los archivos de logs correspondientes. Estos serán para cada ejecución individual y además un archivo general con las últimas líneas devueltas por `cachegrind`.

4. Análisis teórico

La determinación de una predicción teórica, para los resultados a obtener en el comportamiento de la memoria caché, fue uno de los aspectos que se analizó con mayor detenimiento. No solo debieron considerarse los accesos propios de los algoritmos utilizados, sino la disposición en memoria de los elementos de las matrices, su tamaño y el orden de acceso.

Como primer punto, se estudió la ubicación de las matrices en memoria. Un dato a tener en cuenta es que la función `malloc` del lenguaje de programación C devuelve memoria alineada al tamaño especificado. Sin embargo, debido al mapeo en la memoria caché, debe considerarse la distancia en memoria para las matrices. Para los tipos de memoria caché estudiadas, el primer mapeo se realiza con una operación de módulo. Por lo tanto, debe tenerse en cuenta la propiedad:

$$X \pmod{Y} = X + Y \pmod{Y}$$

Lo anterior lleva a considerar que matrices desplazadas Y posiciones en memoria exhibirán el mismo comportamiento con respecto a la ejecución en la memoria caché. Entonces, deben considerarse desplazamientos tales que:

$$\delta = 1, 2, 3, \ldots, Y - 1$$

Considerando que el tipo de dato `double` ocupa 8 bytes en MIPS32, deberán considerarse desplazamientos de:

$$\delta[\text{Bytes}] = 8, 16, 24, \ldots, (Y - 1) * 8$$

donde Y puede tomar los siguientes valores:

- $Y = 1024$ para Mapeo Directo.
- $Y = 512$ para caché asociativo de dos vías.
- $Y = 256$ para caché asociativo de cuatro vías.

En términos simples, se simula primero el algoritmo para matrices contiguas en memoria. Luego, se va desplazando a las matrices en 8 bytes, hasta que la dirección final es equivalente en operador módulo a la matriz inicial (la matriz A).

Al usar matrices de dimensión $\{2^k \mid k \in \{1, 2, 3, 4, 5, 6, 7, 8\}\}$, se tiene garantizado que:

$$2^k \times 8[\text{Bytes}] = 2^{k+3}[\text{Bytes}]$$

y por lo tanto, para bloques de 32 bytes, la cantidad de bloques por matriz será:

$$\text{Bloques} = \frac{2^{k+3}}{32} = 2^{k-2}\text{Bloques}$$

Otro aspecto interesante, es que para un valor de $n = 64$, se tiene:

$$\text{Tamaño en bytes} = n \times n \times 8 = 32768 \tag{1}$$

que es el tamaño de la memoria caché. Por lo tanto, a partir de ese valor, se espera un incremento en el *miss rate*.

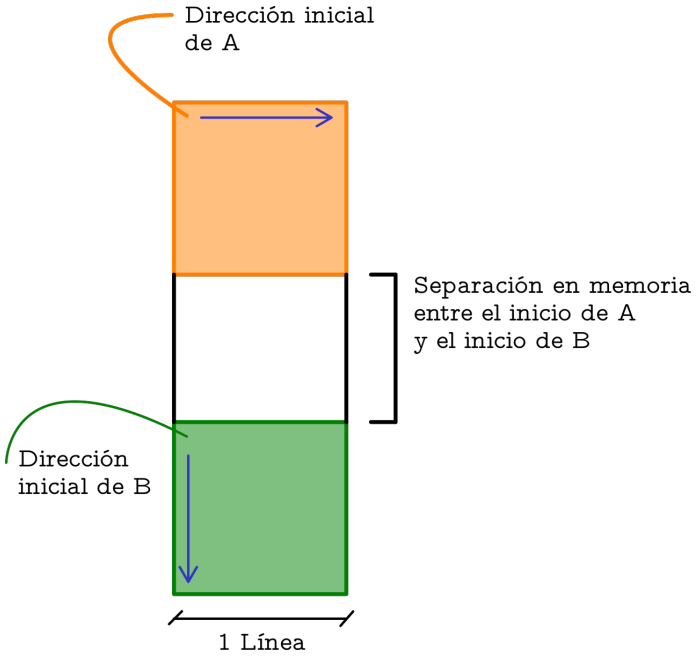


Figura 1: Diagrama que exhibe el desplazamiento en memoria de las matrices utilizadas (la matriz C se omite por simplicidad).

Considerando los desplazamientos anteriores, para las matrices B y C (puede asumirse sin pérdida de generalidad que la matriz A está en una posición arbitraria de la memoria), se realizó una simulación para todos los desplazamientos posibles de memoria, para dimensiones $n = 2^k$. Los resultados obtenidos se exhiben a continuación: En cada figura se muestra la distribución de la tasa de misses para cada dimensión, junto a una línea que denota la media del valor del miss rate para todos los desplazamientos posibles de memoria entre las matrices. Este es el valor teórico contra el cual se compararán las ejecuciones *reales del programa*, para la función programada en assembler.

4.1. Mapeo Directo

Para el caso de mapeo directo, se obtuvo una tasa de miss rate considerablemente alta para dimensiones bajas, lo cual se explica por el *arranque en frío* de la memoria caché: Al tener pocos accesos (dimensión baja) pero a la vez todos los misses obligatorios, estos últimos pesarán en el cálculo del miss rate. El miss rate decrece a medida que aumenta la dimensión, luego crece a partir de la dimensión 32 y vuelve a decrecer a partir de la dimensión 64. Sin embargo, la tasa de miss rate muestra una gran variabilidad, lo cual implica que es muy sensible a la ubicación relativa de las matrices en memoria.

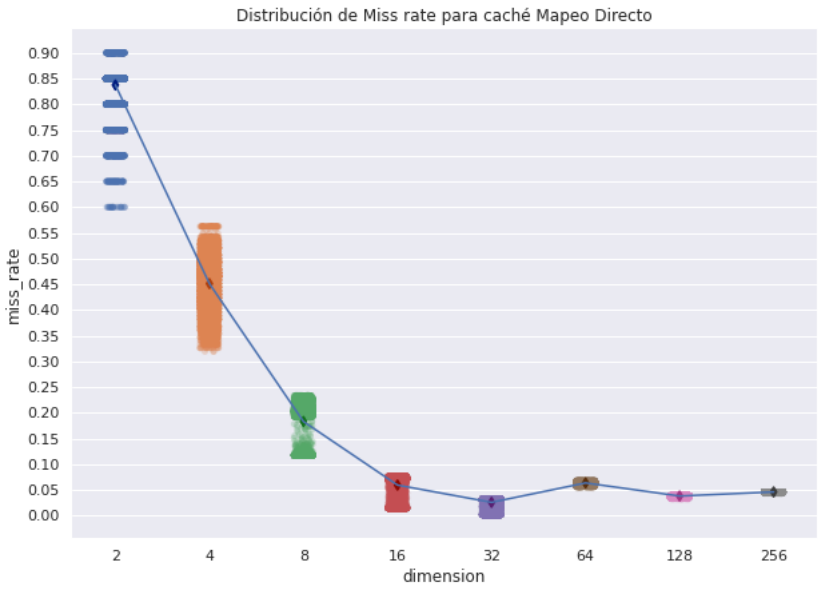


Figura 2: Simulación de miss rate para mapeo directo.

4.2. 2-Way Associative

En la caché asociativa de dos vías, se tiene una menor variabilidad, ya que dentro de cada conjunto, la caché es *fully-associative*. Por iguales motivos a lo expresado en la caché de mapeo directo, la tasa de miss promedio decrece aumentando la dimensión, y luego crece para dimensiones mayores a 32.

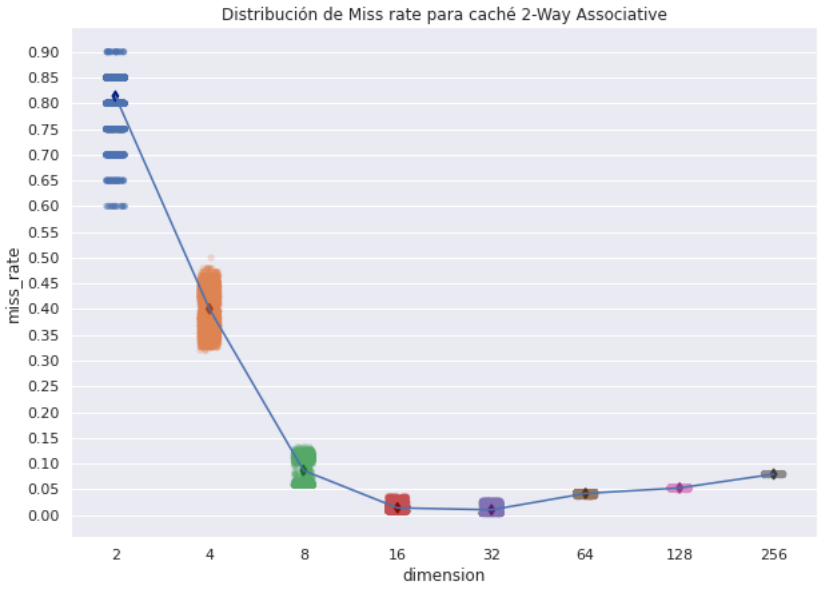


Figura 3: Simulación de miss rate para caché 2-Way associative.

4.3. 4-Way Associative

La memoria caché asociativa de 4 vías es la que exhibe una menor variabilidad: esto tiene sentido ya que direcciones que comparten el índice en la caché tienen menos chances de ser reemplazadas (porque hay más espacio). Nuevamente la tendencia es decreciente, pero se nota un aumento más pronunciado para dimensiones mayores a 32.

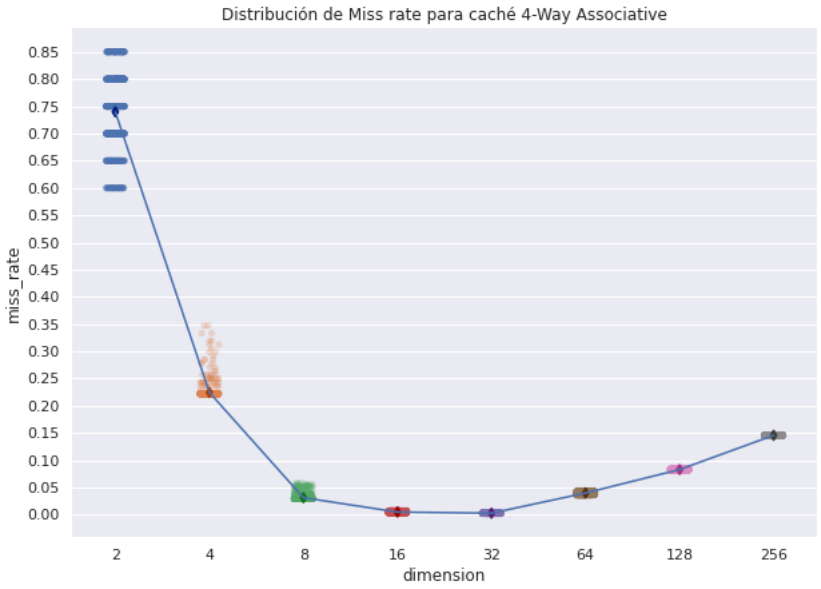


Figura 4: Simulación de miss rate para caché 4-Way associative.

5. Simulación y resultados

5.1. Accesos totales a memoria

La figura siguiente muestra la cantidadad de accesos a memoria según la dimensión. En ella se aprecia la tendencia creciente (polinómica) para los accesos que realiza el algoritmo según la dimensión de las matrices multiplicadas.

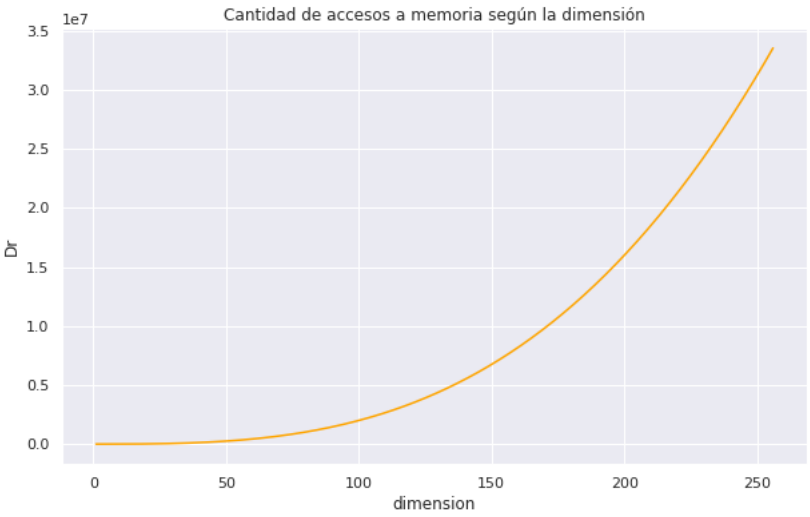


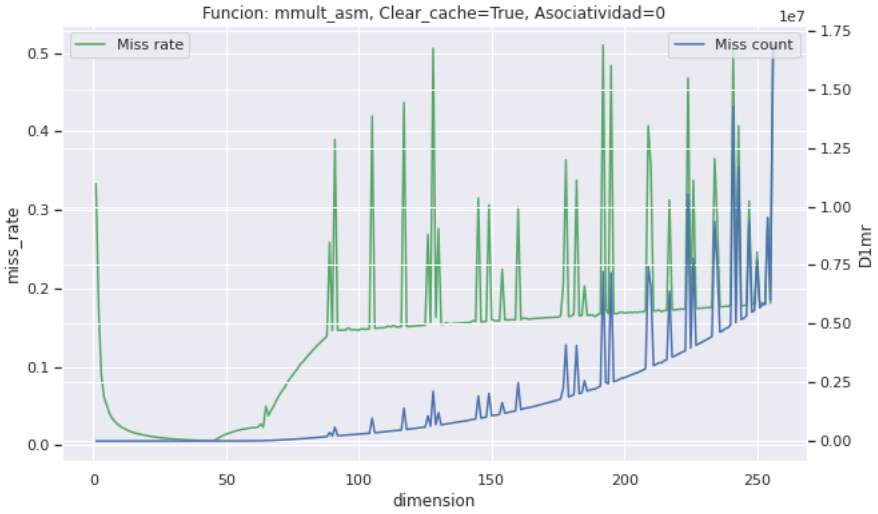
Figura 5: Accesos a memorias según la dimensión.

5.2. Sobre los aciertos en la caché L1D

Si bien las herramientas utilizadas nos permiten determinar la cantidad total de accesos a memoria, así como los desaciertos totales, no nos es posible determinar la cantidad total de aciertos para la cache L1D. Esto se debe a que no podemos diferenciar los desaciertos que pertenecen a dicha memoria cache. La cantidad de aciertos se puede calcular restando los desaciertos de los accesos totales a memoria, pero conociendo los desaciertos de cada nivel no podemos calcular lo pedido.

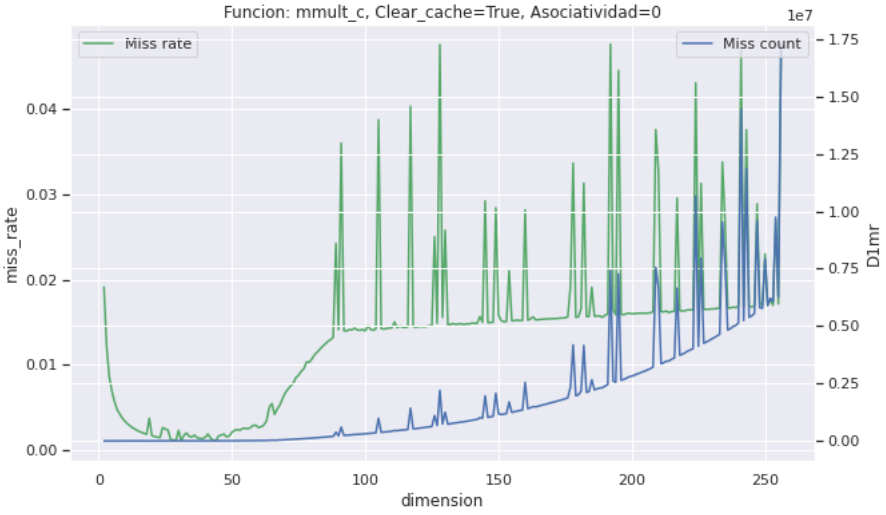
5.3. Mapeo Directo

5.3.1. mmult_asm.S



La cache se encuentra inicialmente vacía, por eso puede apreciarse al comienzo del gráfico valores altos de miss rate, dado que los elementos tienen que moverse a la cache se producen misses forzosos. Dado la baja cantidad de accesos a memoria que se hacen por las bajas dimensiones de las matrices multiplicadas, estos misses tienen una mayor importancia, lo que genera la alta tasa de desaciertos. Tal como se esperaba, alrededor de la dimensión 64 se produce un incremento de la tasa de desaciertos.

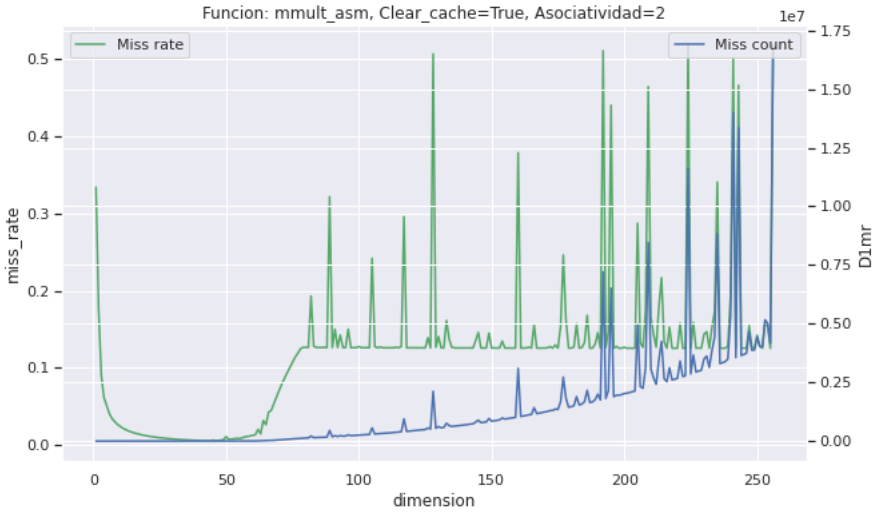
5.3.2. mmult_naive_c.c



El caso de la función usada provista por la cátedra es similar al caso anterior. Se produce un pico en la tasa de desaciertos para dimensiones bajas, dado que el numero de desaciertos tiene más peso en comparación con la cantidad total de accesos. Puede observarse que la tasa de desaciertos es más chica utilizando esta implementación de la multiplicación. Esto se debe a que se realiza una mayor cantidad de accesos a memoria en esta ejecución, disminuyendo la tasa de desaciertos. Se observa nuevamente, un incremento en la tasa de desaciertos para las dimensiones cercanas a 64.

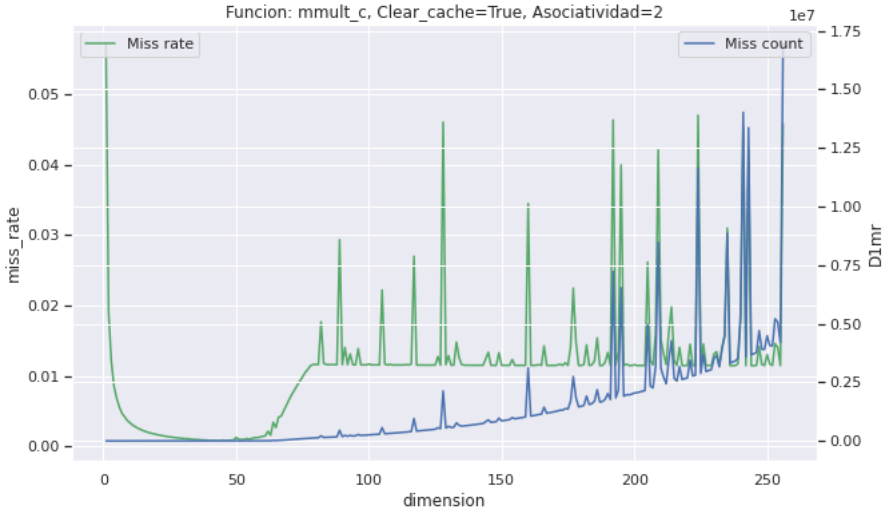
5.4. 2-Way Associative

5.4.1. mmult_asm.S



Se observa el pico al comienzo del gráfico propio de haber realizado un arranque en frío del cache, así como el incremento en la dimensión 64. Puede notarse un decremento en el 'piso' de la tasa de desaciertos, en comparación con el gráfico que implementa la misma función en un cache direct mapped. Esto es de esperarse, dado que incrementar la asociatividad reduce los posibles conflictos que puede producirse al llevar direcciones de memoria a cache.

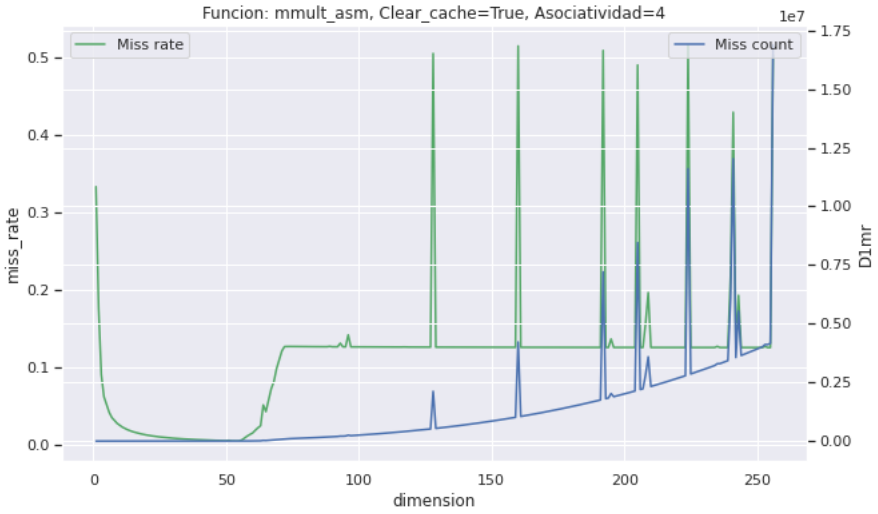
5.4.2. mmult_naive_c.c



Puede observarse el efecto de haber incrementado la asociatividad en este gráfico, habiendo disminuido la tasa de desaciertos en escala en comparación con la version de cache direct mapped. La disminución en la cantidad de picos producidos en la tasa de desaciertos se le puede atribuir a que disminuyeron los casos de trashing.

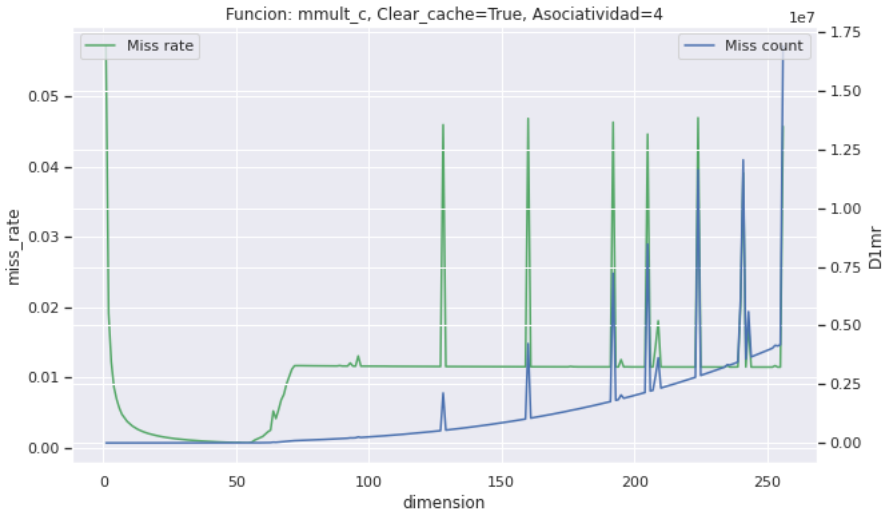
5.5. 4-Way Associative

5.5.1. mmult_asm.S



Al igual que en configuraciones anteriores, se observa un miss rate elevado para dimensiones muy pequeñas. Por otro lado, se tienen menos casos de *trashing* severo, conforme con el aumento de la asociatividad. El miss rate muestra valores similares a los obtenidos para la misma función en la caché de dos vías.

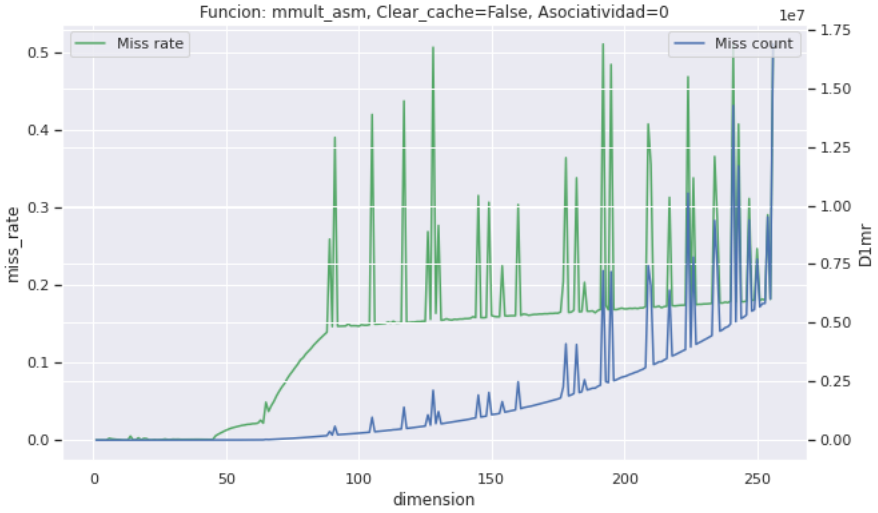
5.5.2. mmult_naive_c.c

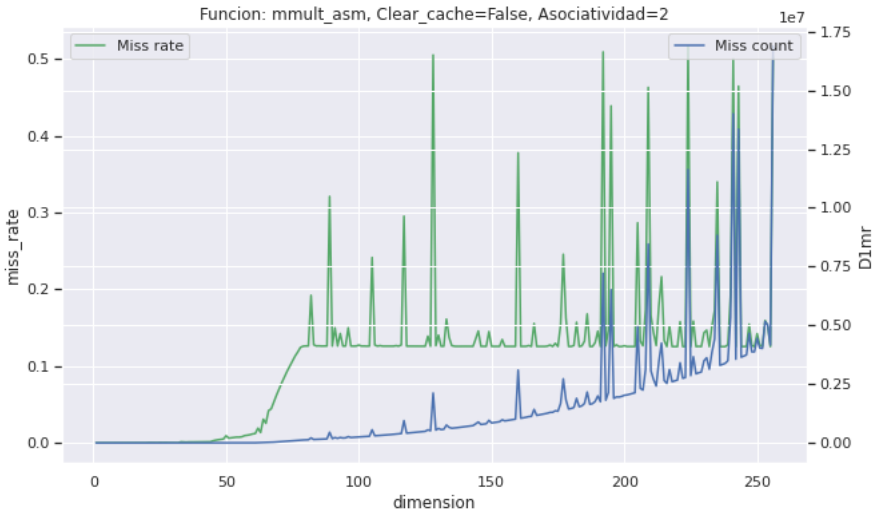


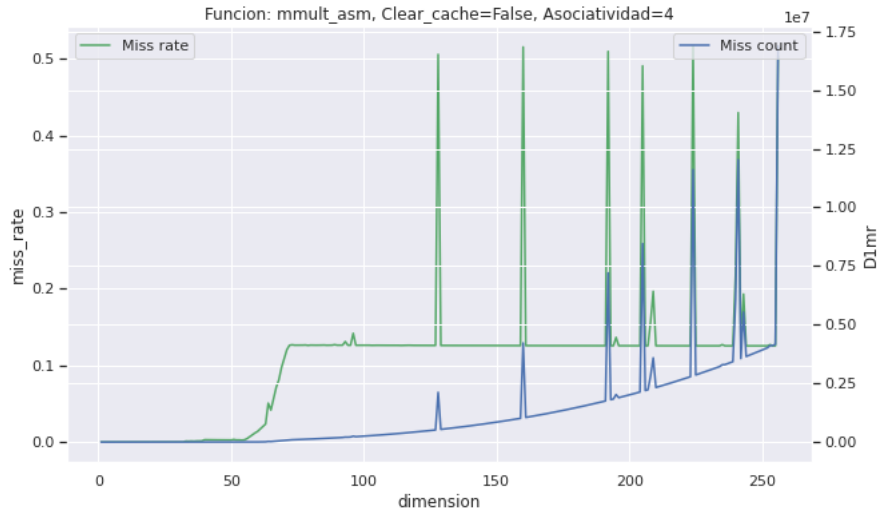
Para la función no optimizada de multiplicación, se tiene un resultado similar, salvando en el inicio: los accesos repetidos a la matriz C tienen un peso mayor en dimensiones pequeñas.

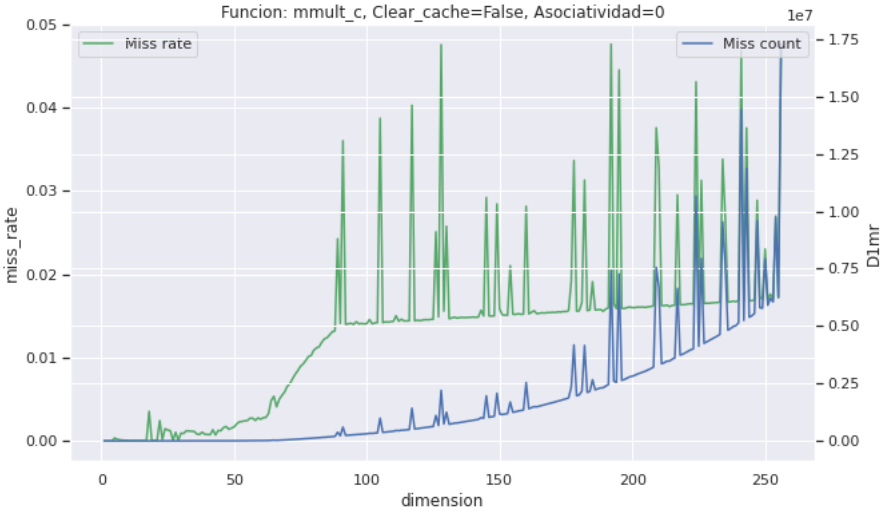
5.6. Análisis sin cold-start

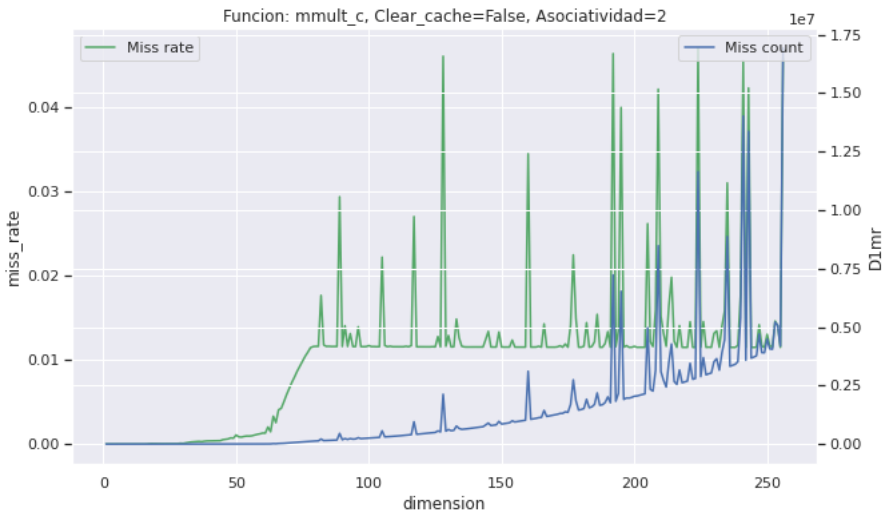
En esta sección se muestran análisis relacionados, para casos en los que no se *reseteó* la memoria caché antes de realizar la multiplicación. En contraposición a las corridas anteriores, se observa un miss rate prácticamente despreciable para dimensiones pequeñas, ya que la localidad temporal entra en juego. Como se esperaba, se produce un incremento de la tasa de desaciertos alrededor de la dimensión 64. En todos los casos la tasa de miss rate mejora con la asociatividad, al igual que se produce una menor cantidad de casos de *trashing*.

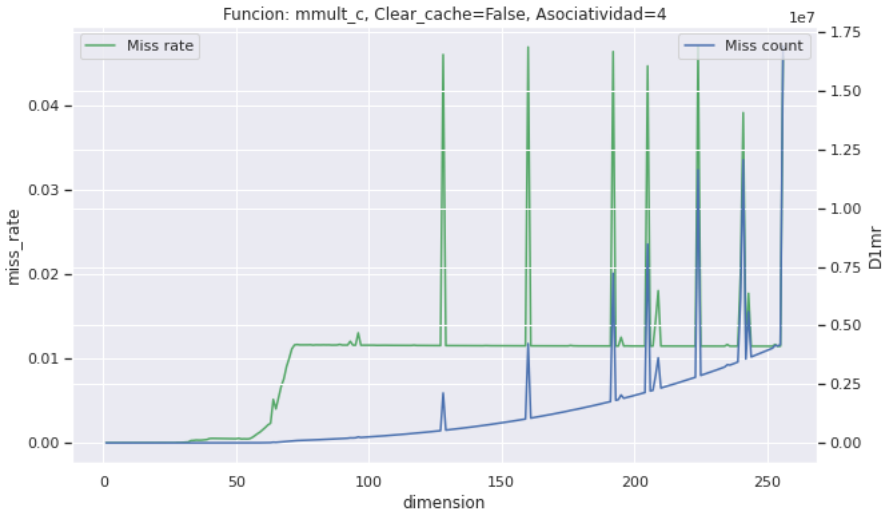

Funcion: mmult_asm, Clear_cache=False, Asociatividad=0
Miss rate
Miss count
1e7
miss_rate
D1mr
dimension


Funcion: mmult_asm, Clear_cache=False, Asociatividad=2
Miss rate
Miss count
1e7
miss_rate
D1mr
dimension


Funcion: mmult_asm, Clear_cache=False, Asociatividad=4
1e7
Miss rate
Miss count
miss_rate
D1mr
dimension
0.0
0.1
0.2
0.3
0.4
0.5
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
50
100
150
200
250


Funcion: mmult_c, Clear_cache=False, Asociatividad=0
1e7
Miss rate
Miss count
miss_rate
D1mr
dimension
0.00
0.01
0.02
0.03
0.04
0.05
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
0
50
100
150
200
250

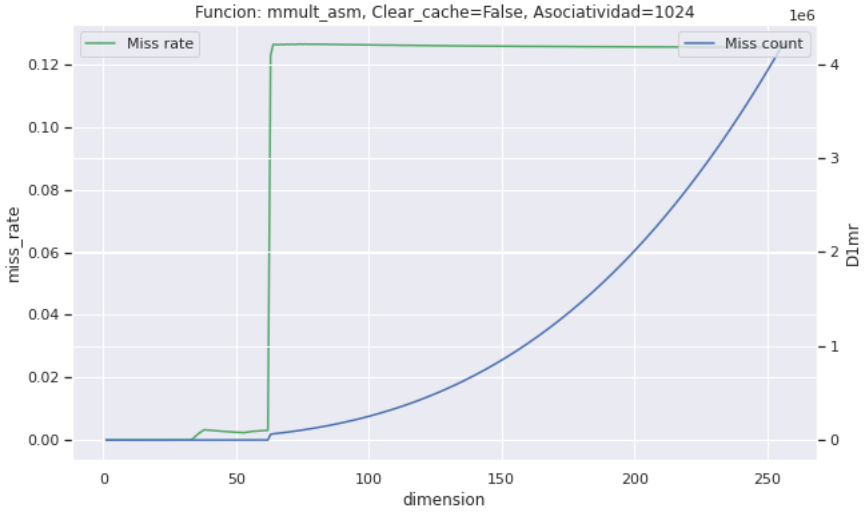


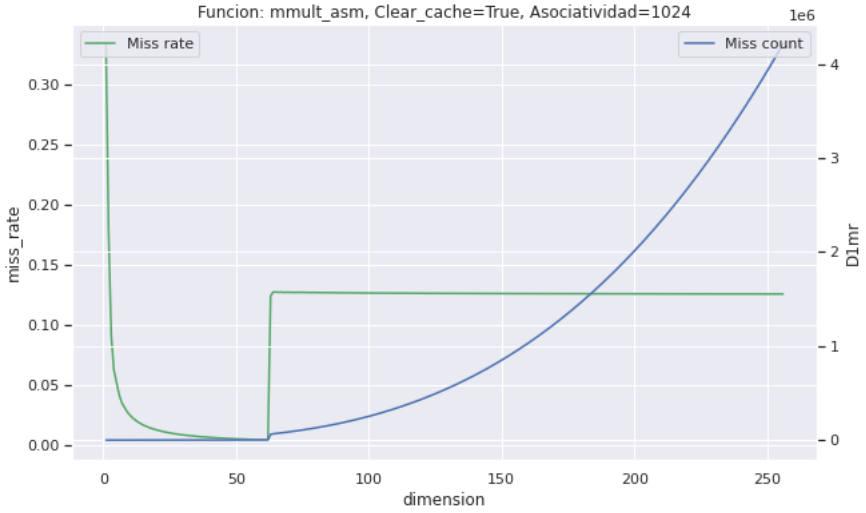


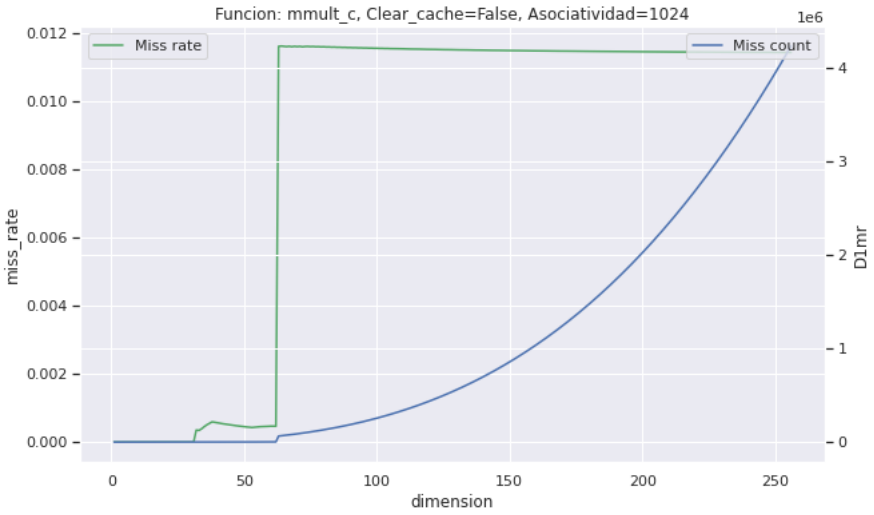
5.7. Fully Associative

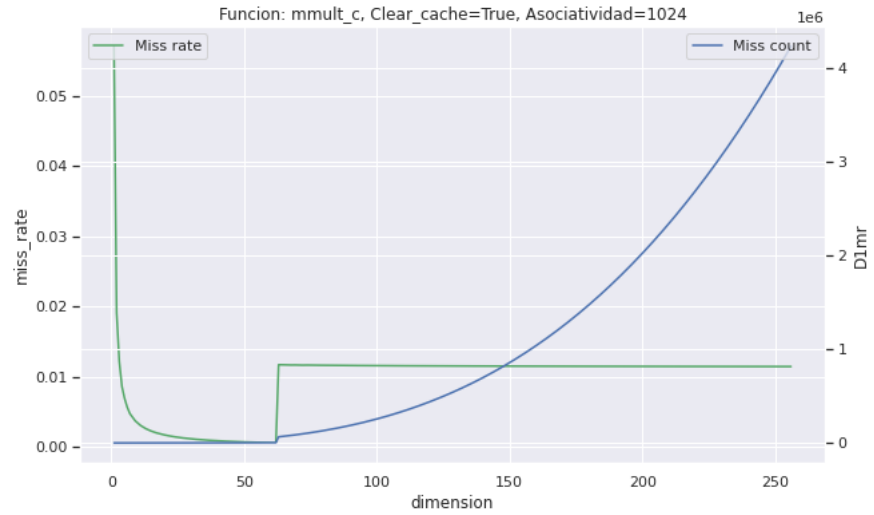
En esta sección se muestran los resultados obtenidos para corridas con memoria caché *Fully-associative*. Dependiendo de si se borró la caché o no antes de realizar una multiplicación, el miss rate tendrá valores muy elevados o despreciables, respectivamente. También debe notarse que la tasa de miss es mucho menor que en las demás configuraciones, y no se producen casos

de trashing severo en ningún caso. El miss rate se dispara, nuevamente, para matrices de dimensión mayor a 64.









6. Conclusiones

Las conclusiones que se exponen a continuación son un contraste entre los resultados obtenidos con `cachegrind` y las simulaciónes de caché realizadas, para la función programada en assembler (ubicada en `src/mult/mmult_asm.S`. Los resultados siguientes dan por aludido que es ese algoritmo el que se está utilizando.

6.1. Mapeo Directo

Para el caso de mapeo directo, se observó que las predicciones no coincidieron con lo esperado, sino que se tuvo un miss-rate algo mayor. La diferencia observada se atribuye a que las tres matrices a multiplicar se encuentran contiguas en memoria (un solo `malloc`. (Los resultados se comparan con los obtenidos para la función programada en assembler, la función en C realiza más accesos).

6.2. 2-Way Associative

En el caso de caché 2-Way associative, se observan resultados similares a los esperados teóricamente, y mejores a los de la caché de Mapeo Directo. Aumentar las vías hace que se puedan resolver más colisiones al calcular el módulo de la dirección de memoria.

6.3. 4-Way Associative

Para el caso de la caché asociativa de 4 vías, los resultados obtenidos coinciden también con lo esperado teóricamente. Sin embargo, no se observa un crecimiento en el miss rate, que puede atribuirse nuevamente a la contiguidad de las matrices en memoria, o bien a algún error en la inferencia teórica del miss-rate a obtener.

Apéndices

A. Repositorio del grupo

Repositorio del trabajo practico donde se puede encontrar todo el código y archivos de entrada y salida utilizados para su realización:
`https://github.com/mateoicalvo/6620/tp2`

B. Anotaciones del CG_anotate

A continuación se encuentra el archivo producido por CG_annotate para la ejecución del programa con matrices de tamaño 60 con una configuración direct-mapped. Se pueden encontrar los demás archivos generados para las distintas configuraciones de cache y dimensiones de matrices en el repositorio del grupo.

```
--------------------------------------------------------------------------------
I1 cache:         32768 B, 32 B, 4-way associative
D1 cache:         32768 B, 32 B, direct-mapped
LL cache:         524288 B, 32 B, 8-way associative
Command:          bin/benchmark
Data file:        cachegrind.out.911
Events recorded:  Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Events shown:     Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Event sort order: Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
Thresholds:       0.1 100 100 100 100 100 100 100 100
Include dirs:
User annotated:   /root/tp2/asm/mmult.s
Auto-annotation:  off

--------------------------------------------------------------------------------
Ir          I1mr  ILmr  Dr         D1mr   DLmr  Dw         D1mw      DLmw
--------------------------------------------------------------------------------
169,355,216 2,510 2,489 60,076,338 38,115 5,754 22,924,211 1,327,975
1,314,491  PROGRAM TOTALS

--------------------------------------------------------------------------------
Ir          I1mr ILmr Dr         D1mr   DLmr  Dw         D1mw      DLmw
file:function
--------------------------------------------------------------------------------
146,800,874   32   32 52,428,873 10,257     7 20,971,539 1,319,682
1,310,720  /root/tp2/src/benchmark.c:benchmark_correr
  9,994,581   11   11  4,770,303 12,653 2,701    439,326         0
0  /root/tp2/asm/mmult.s:mmult
  2,813,903  123  123    735,195  7,433     8    260,715     1,075
78  /build/glibc-qeih7b/glibc-2.24/stdio-common/printf_fp.c:__printf_fp_l
  1,908,000   51   51    338,400    931     1    201,600        92
0  /build/glibc-qeih7b/glibc-2.24/stdlib/strtod_l.c:____strtod_l_internal
  1,067,837   16   16    225,633      0     0    196,833       108
0  /build/glibc-qeih7b/glibc-2.24/stdlib/divrem.c:__mpn_divrem
  1,017,199   65   65    301,820    401    26    174,347       198
2  /build/glibc-qeih7b/glibc-2.24/stdio-common/vfprintf.c:vfprintf
    898,912   15   15    284,192      1     0    142,476        51
0  /build/glibc-qeih7b/glibc-2.24/stdio-common/printf_fp.c:hack_digit
    532,800   12   12    158,400      9     0     86,400        26
0  /build/glibc-qeih7b/glibc-2.24/stdlib/strtod_l.c:round_and_return
    525,600   13   13    129,600    169     1    100,800        78
0  /build/glibc-qeih7b/glibc-2.24/stdlib/strtod_l.c:str_to_mpn.isra.0
    407,300   39   39     32,643     20     5     32,617     1,178
514  /build/glibc-qeih7b/glibc-
2.24/string/../sysdeps/mips/memcpy.S:memcpy
    341,518   12   12     72,674     69     3     65,382         2
2  /build/glibc-qeih7b/glibc-
2.24/libio/fileops.c:_IO_file_xsputn@@GLIBC_2.2
    290,692    9    9     21,802      6     2          0         0
0  /build/glibc-qeih7b/glibc-2.24/string/strchrnul.c:strchrnul
    273,680   11   11    115,228     85     0     28,820     1,988
0  /root/tp2/src/matriz.c:matriz_parsear
```

```
    260,150   12   12     32,505     57     4          0         0
0  /build/glibc-qeih7b/glibc-2.24/string/strlen.c:strlen
    223,228    5    5     14,402      0     0     14,402        38
0  /build/glibc-qeih7b/glibc-
2.24/stdlib/../sysdeps/mips/lshift.S:__mpn_lshift
    207,252    5    5     14,369      0     0     14,369         0
0  /build/glibc-qeih7b/glibc-
2.24/stdlib/../sysdeps/mips/mul_1.S:__mpn_mul_1
    180,035    4    4     14,403      0     0     14,403         0
0  /build/glibc-qeih7b/glibc-
2.24/stdlib/../sysdeps/mips/rshift.S:__mpn_rshift

--------------------------------------------------------------------------------
-------
-- User-annotated source: /root/tp2/asm/mmult.s
--------------------------------------------------------------------------------
-------
Ir        I1mr ILmr Dr        D1mr   DLmr  Dw      D1mw DLmw

        .    .    .         .      .     .       .    .    .   .section
.mdebug.abi32
        .    .    .         .      .     .       .    .    .   .previous
        .    .    .         .      .     .       .    .    .   .nan
      legacy
        .    .    .         .      .     .       .    .    .   .module
      fp=xx
        .    .    .         .      .     .       .    .    .   .module
      nooddspreg
      488    2    2       182      0     0      61    0    0   .abicalls
   29,280    2    2    10,920      0     0   3,660    0    0   .text
1,756,800    1    1   655,200    240     0 219,600    0    0  $Ltext0:
4,104,000    2    2 1,944,000  1,141   901 216,000    0    0
      .cfi_sections    .debug_frame
4,104,000    2    2 2,160,000 11,272 1,800       0    0    0   .align
      2
        .    .    .         .      .     .       .    .    .   .globl
      mmult
        .    .    .         .      .     .       .    .    .  $LFB0 = .
        .    .    .         .      .     .       .    .    .
      .cfi_startproc
       16    2    2         1      0     0       5    0    0   .set
      nomips16
        .    .    .         .      .     .       .    .    .   .set
      nomicromips
        .    .    .         .      .     .       .    .    .   .ent
      mmult
        .    .    .         .      .     .       .    .    .   .type
      mmult, @function
        .    .    .         .      .     .       .    .    .  mmult:
        .    .    .         .      .     .       .    .    .   .frame
      $fp,32,$31       # vars= 16, regs= 1/0, args= 0, gp= 8
        .    .    .         .      .     .       .    .    .   .mask
      0x40000000,-4
        .    .    .         .      .     .       .    .    .   .fmask
      0x00000000,0
        .    .    .         .      .     .       .    .    .   .set
      noreorder
```

```
          .    .    .          .      .     .       .    .    .   .set
      nomacro
          1    1    1          0      0     0       0    0    0   addiu
      $sp,$sp,-32
          .    .    .          .      .     .       .    .    .
      .cfi_def_cfa_offset 32
          1    0    0          0      0     0       1    0    0   sw
      $fp,28($sp)
          .    .    .          .      .     .       .    .    .
      .cfi_offset 30, -4
          1    0    0          0      0     0       0    0    0   move
      $fp,$sp
          .    .    .          .      .     .       .    .    .
      .cfi_def_cfa_register 30
          1    0    0          0      0     0       1    0    0   sw
      $4,32($fp)
          1    1    1          0      0     0       1    0    0   sw
      $5,36($fp)
          1    0    0          0      0     0       1    0    0   sw
      $6,40($fp)
          1    0    0          0      0     0       1    0    0   sw
      $7,44($fp)
          .    .    .          .      .     .       .    .    .   $LBB2 = .
          .    .    .          .      .     .       .    .    .   .loc 1 6 0
          .    .    .          .      .     .       .    .    .   sw
      $0,8($fp)
          .    .    .          .      .     .       .    .    .   b     $L2
          .    .    .          .      .     .       .    .    .   nop
          .    .    .          .      .     .       .    .    .
          .    .    .          .      .     .       .    .    .   $L7:
          .    .    .          .      .     .       .    .    .   $LBB3 = .
-- line 41 ----------------------------------------

--------------------------------------------------------------------------------
-------
Ir         I1mr ILmr Dr         D1mr    DLmr   Dw       D1mw DLmw
--------------------------------------------------------------------------------
-------
9,994,591   13   13 4,770,303 12,653 2,701 439,331    0    0  events
annotated
```

C. Código fuente

```c
#include "benchmark.h"

#include <stddef.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>

#include "matriz.h"
#include "matriz_helpers.h"
#include "cronometro.h"

#define CLEAR_CACHE 1

int parsear_dimension(benchmark_t* benchmark, size_t* dimension) {
    size_t parseada = strtol(benchmark->matrices, \
            &(benchmark->cursor), 10);

    if (errno) {
            perror("");
                return BENCHMARK_ERROR_DIMENSION;
        }
        if (benchmark->matrices == benchmark->cursor) {
                fprintf(stderr, MSG_BENCHMARK_ERROR_FALTA_DIMENSION);
                return BENCHMARK_ERROR_DIMENSION;
        }
        if (parseada == 0) {
                fprintf(stderr, MSG_BENCHMARK_ERROR_DIMENSION);
                return BENCHMARK_ERROR_DIMENSION;
        }
    *dimension = parseada;
    return BENCHMARK_DIMENSION_OK;
}

int crear_matrices(benchmark_t* benchmark, matriz_t* A, matriz_t* B, \
    matriz_t* C, double* datos, size_t dimension) {

    size_t tamanio_matriz = dimension*dimension;
    matriz_crear_desde(A, datos, dimension);
    matriz_crear_desde(B, datos+tamanio_matriz, dimension);
    matriz_crear_desde(C, datos+tamanio_matriz*2, dimension);

    int resultado_A = matriz_parsear(A, &benchmark->cursor);
    if (resultado_A == MATRIZ_ERROR) {
        goto destruirA;
    }
    int resultado_B = matriz_parsear(B, &benchmark->cursor);
    if (resultado_B == MATRIZ_ERROR) {
        goto destruirB;
    }
    return MATRIZ_OK;

    destruirB:
    matriz_destruir(B);
    destruirA:
    matriz_destruir(A);

    matriz_destruir(C);
    return MATRIZ_ERROR;
}

void destruir_matrices(matriz_t* A, matriz_t* B, \
    matriz_t* C) {
        matriz_destruir(A);
        matriz_destruir(B);
        matriz_destruir(C);
}


```

```
 69: void benchmark_crear(benchmark_t* benchmark) {
 70:     benchmark->matrices = NULL;
 71:     benchmark->cursor = " ";
 72: }
 73:
 74: int benchmark_correr(benchmark_t* benchmark) {
 75:     matriz_t A;
 76:     matriz_t B;
 77:     matriz_t C;
 78:
 79:     cronometro_t cronometro;
 80:     cronometro_crear(&cronometro);
 81:
 82:     size_t tamanio = 0;
 83:     int resultado = 0;
 84:     while ((resultado = getline(&(benchmark->matrices), &tamanio, \
 85:         stdin)) != -1) {
 86:
 87:         size_t dimension;
 88:         int resultado = parsear_dimension(benchmark, &dimension);
 89:         if (resultado == BENCHMARK_ERROR_DIMENSION) {
 90:             goto fin_error;
 91:         }
 92:
 93:         size_t total_elementos = sizeof(double)*dimension*dimension*3;
 94:         double* datos = malloc(total_elementos);
 95:         if (!datos) {
 96:             return BENCHMARK_RESULTADO_ERROR;
 97:         }
 98:
 99:         resultado = crear_matrices(benchmark, &A, &B, &C, datos, dimension);
100:         if(resultado == MATRIZ_ERROR){
101:             free(datos);
102:             return BENCHMARK_RESULTADO_ERROR;
103:         }
104:         cronometro_iniciar(&cronometro);
105:
106:         #if CLEAR_CACHE
107:         size_t j;
108:                 size_t dim = 1024*1024*10;
109:                 int *v = malloc(dim*sizeof(int));
110:         if(!v) {
111:             free(datos);
112:             destruir_matrices(&A, &B, &C);
113:             return BENCHMARK_RESULTADO_ERROR;
114:         }
115:                 for (j = 0; j < dim; ++j)
116:                         v[j] = -1;
117:                 free(v);
118:         #endif
119:
120:         matriz_multiplicar(&A, &B, &C, &mmult);
121:
122:         cronometro_detener(&cronometro);
123:
124:         matriz_imprimir(&C, stdout);
125:         cronometro_log(&cronometro);
126:
127:         destruir_matrices(&A, &B, &C);
128:         free(datos);
129:         free(benchmark->matrices);
130:         benchmark->matrices = NULL;
131:     }
132:     if (feof(stdin)) {
133:         resultado = BENCHMARK_RESULTADO_OK;
134:     } else {
135:         resultado = BENCHMARK_RESULTADO_ERROR;
136:     }
```

```
137:     free(benchmark->matrices);
138:     return resultado;
139: fin_error:
140:     free(benchmark->matrices);
141:     return BENCHMARK_RESULTADO_ERROR;
142: }
143:
144: void benchmark_destruir(benchmark_t* benchmark) {
145:
146: }
```

```
 1: #ifndef BENCHMARK_H
 2: #define BENCHMARK_H
 3:
 4: #define BENCHMARK_RESULTADO_OK 0
 5: #define BENCHMARK_RESULTADO_ERROR -1
 6:
 7: #define BENCHMARK_ERROR_DIMENSION -1
 8: #define BENCHMARK_DIMENSION_OK 0
 9:
10: #define MSG_BENCHMARK_ERROR_FALTA_DIMENSION "\
11: Error, falta la dimension.\n"
12:
13: #define MSG_BENCHMARK_ERROR_DIMENSION "\
14: Error, dimension invalida.\n"
15:
16:
17: typedef struct benchmark {
18:     char* matrices;
19:     char* cursor;
20: } benchmark_t;
21:
22: void benchmark_crear(benchmark_t* benchmark);
23:
24: int benchmark_correr(benchmark_t* benchmark);
25:
26: void benchmark_destruir(benchmark_t* benchmark);
27:
28: #endif
```

```
 1: #include "cronometro.h"
 2:
 3: #include <stdio.h>
 4:
 5: void cronometro_crear(cronometro_t* cronometro) {
 6: }
 7:
 8: void cronometro_iniciar(cronometro_t* cronometro) {
 9:     clock_gettime(CLOCK_REALTIME, &(cronometro->t_inicial));
10: }
11:
12: void cronometro_detener(cronometro_t* cronometro) {
13:     clock_gettime(CLOCK_REALTIME, &(cronometro->t_final));
14: }
15:
16: double cronometro_tiempo_transcurrido(cronometro_t* cronometro) {
17:
18:     double segundos = (float) cronometro->t_final.tv_sec \
19:          - cronometro->t_inicial.tv_sec;
20:     double nanosegundos = ((float) cronometro->t_final.tv_nsec \
21:          - cronometro->t_inicial.tv_nsec) / 1.0e9;
22:
23:     return segundos + nanosegundos;
24: }
25:
26: void cronometro_log(cronometro_t* cronometro) {
27:     double t = cronometro_tiempo_transcurrido(cronometro);
28:     fprintf(stdout, "Tiempo trasncurrido: %g\n", t);
29: }
30:
31: void cronometro_destruir(cronometro_t* cronometro) {
32: }
```

```
 1: #ifndef CRONOMETRO_H
 2: #define CRONOMETRO_H
 3:
 4: #include <time.h>
 5:
 6: typedef struct cronometro {
 7:     struct timespec t_inicial;
 8:         struct timespec t_final;
 9: } cronometro_t;
10:
11:
12: void cronometro_crear(cronometro_t* cronometro);
13:
14: void cronometro_iniciar(cronometro_t* cronometro);
15:
16: void cronometro_detener(cronometro_t* cronometro);
17:
18: double cronometro_tiempo_transcurrido(cronometro_t* cronometro);
19:
20: void cronometro_destruir(cronometro_t* cronometro);
21:
22: void cronometro_log(cronometro_t* cronometro);
23:
24: #endif
```

```
#include "matriz.h"

#include <stdlib.h>
#include <errno.h>

int matriz_crear(matriz_t* matriz, size_t dimension, bool borrar_datos) {
    matriz->elementos = malloc(sizeof(double)*dimension*dimension);
    if (!matriz->elementos) {
        fprintf(stderr, MSG_MATRIZ_ERROR_MALLOC);
        return MATRIZ_ERROR_CREAR;
    }
    size_t cantidad_elementos = dimension * dimension;
    for (size_t elemento = 0; elemento < cantidad_elementos;\
        elemento++) {
            matriz->elementos[elemento] = 0.0;
    }
    matriz->dimension = dimension;
    matriz->borrar_datos = borrar_datos;
    return MATRIZ_OK;
}

void matriz_crear_desde(matriz_t* matriz, double* elementos, size_t dimension) {
    matriz->elementos = elementos;
    size_t cantidad_elementos = dimension * dimension;
    for (size_t elemento = 0; elemento < cantidad_elementos;\
        elemento++) {
            matriz->elementos[elemento] = 0.0;
    }
    matriz->dimension = dimension;
    matriz->borrar_datos = false;
}


int matriz_parsear(matriz_t* matriz, char** elementos) {

    char* nptr = *elementos;
    char* endptr = *elementos;
    size_t n = matriz->dimension * matriz->dimension;
    double elemento;
    for (size_t i = 0; i < n; i++) {
        nptr = endptr;
        elemento = strtod(nptr, &endptr);
                if (errno) {
                        perror("");
            return MATRIZ_ERROR;
                }
                if (nptr == endptr) {
                        fprintf(stderr, MSG_MATRIZ_ERROR_PARSEAR);
                }
        matriz->elementos[i] = elemento;
    }
    *elementos = endptr;
    return MATRIZ_OK;
}

void matriz_destruir(matriz_t* matriz) {
    if (matriz->elementos != NULL && matriz->borrar_datos) {
        free(matriz->elementos);
    }
    matriz->elementos = NULL;
}

int matriz_imprimir(matriz_t* matriz, FILE* destino) {

    int resultado = fprintf(destino, "Dimension: %zu.\n",
        matriz->dimension);
    if (resultado < 0) {
        perror("");
```

```
   69:         return MATRIZ_ERROR;
   70:     }
   71:
   72:     size_t dimension = matriz->dimension;
   73:     for(size_t fila = 0; fila < matriz->dimension; fila++) {
   74:         for(size_t columna = 0; columna < matriz->dimension;\
   75:             columna++) {
   76:
   77:                 resultado = fprintf(destino, " %g ", \
   78:                     matriz->elementos[fila*dimension + columna]);
   79:                 if (resultado < 0) {
   80:                     perror("");
   81:                     return MATRIZ_ERROR;
   82:                 }
   83:         }
   84:         resultado = fprintf(destino, "\n");
   85:         if (resultado < 0) {
   86:             perror("");
   87:             return MATRIZ_ERROR;
   88:         }
   89:     }
   90:         return MATRIZ_OK;
   91: }
   92:
   93: void matriz_multiplicar(matriz_t* A, matriz_t* B,
   94:     matriz_t* resultado,
   95:     void (*f)(double*, double*, double*, size_t)) {
   96:
   97:     double* elementosA = A->elementos;
   98:     double* elementosB = B->elementos;
   99:     double* elementosC = resultado->elementos;
  100:
  101:     size_t n = A->dimension;
  102:
  103:     f(elementosA, elementosB,\
  104:         elementosC, n);
  105: }
```

```
 1: #ifndef MATRIZ_H
 2: #define MATRIZ_H
 3:
 4: #include <stddef.h>
 5: #include <stdio.h>
 6: #include <stdbool.h>
 7:
 8: #define MSG_MATRIZ_ERROR_MALLOC "\
 9: Error matriz_crear, fallo malloc\n"
10:
11: #define MSG_MATRIZ_ERROR_PARSEAR "\
12: Error matriz_parsear, faltan elementos\n"
13:
14: #define MATRIZ_ERROR_CREAR -1
15: #define MATRIZ_OK 0
16: #define MATRIZ_ERROR -1
17:
18: typedef struct matriz {
19:         size_t dimension;
20:         double* elementos;
21:         bool borrar_datos;
22: } matriz_t;
23:
24: int matriz_crear(matriz_t* matriz, size_t dimension, bool borrar_datos);
25:
26: void matriz_crear_desde(matriz_t* matriz, double* elementos, size_t dimension);
27:
28: int matriz_parsear(matriz_t* matriz, char** elementos);
29:
30: void matriz_destruir(matriz_t* matriz);
31:
32: int matriz_imprimir(matriz_t* matriz, FILE* destino);
33:
34: void matriz_multiplicar(matriz_t* A, matriz_t* B,
35:     matriz_t* resultado, \
36:         void (*f)(double*, double*, double*, size_t));
37:
38: #endif
```

```
1: #include <stddef.h>
2:
3: void mmult(double* A, double* B, double* resultado, size_t n);
```

```
 1: #include <stddef.h>
 2:
 3: void mmult(double* A, double* B, \
 4:     double* resultado, size_t n) {
 5:
 6:         for(size_t fila = 0; fila < n; fila++) {
 7:             for(size_t columna = 0; columna < n; columna++) {
 8:                 for(size_t i = 0; i < n; i++) {
 9:                     resultado[fila*n + columna] += \
10:                         (A[fila*n + i] * B[i*n + columna]);
11:                 }
12:             }
13:         }
14: }
```

```c
#include <stdio.h>
#include "benchmark.h"

#define MSG_ERROR_BENCHMARK "Error en la ejecucion del benchmark.\n"

int main(int argc, char const *argv[]) {

    benchmark_t benchmark;
    benchmark_crear(&benchmark);
    int resultado = benchmark_correr(&benchmark);
    if (resultado == BENCHMARK_RESULTADO_ERROR) {
        fprintf(stderr, MSG_ERROR_BENCHMARK);
    }

    benchmark_destruir(&benchmark);
    return resultado;
}
```

```
 1: #include <sys/regdef.h>
 2:
 3:     .abicalls
 4:     .text
 5:     .align 2
 6:     .set oddspreg
 7:     .globl mmult
 8:     .ent mmult
 9:
10: mmult:
11:     .frame fp, 8, ra
12:     .set noreorder
13:     .cpload t9
14:     .set reorder
15:
16:     subu sp, sp, 8
17:     .cprestore 0
18:         sw fp, 4(sp)
19:     move fp, sp
20:
21:     #En el ABA de la caller
22:     sw a0, 8(fp)  #matriz_t* A esta en fp+8
23:         sw a1, 12(fp) #matriz_t* B esta en fp+12
24:         sw a2, 16(fp) #matriz_t* C esta en fp+16
25:     sw a3, 20(fp) #dimension (n) esta en fp+20
26:
27:     li t3, 0   #t3 = 0 (fila)
28: iter_filas:
29:     beq t3, a3, salir #fila == n?
30:     li t4, 0           #t4 = 0 (columna)
31:     iter_columnas:
32:         beq t4, a3, inc_fila    #columna == n?
33:         li t5, 0                   #t5 = 0 (i)
34:         mtc1 zero, $f6 #f6 vale cero, aca guardo sumas parciales
35:         mtc1 zero, $f7 #f7 seteado a cero porque trabajo con doubles
36:         iter_n:
37:             beq t5, a3, inc_columna #i == n? -> columna ++
38:             mul t6, a3, t3  #t6 = fila * n
39:             add t6, t6, t5  #t6 es fila*n + i, t6 es indice de A
40:
41:             sll t0, t6, 3   #t0 = t6 * 8
42:             add t0, a0, t0  #t0 tiene la direccion de A[t6]
43:             ldc1 $f0, 0(t0) #f0 tiene el elemento de A
44:
45:             mul t7, t5, a3  #t7 = i * n
46:             add t7, t7, t4  #t7 es i*n + columna, t7 es indice de B
47:
48:             sll t1, t7, 3   #t1 = t7 *8
49:             add t1, a1, t1  #t1 tiene la direccion de B[t6]
50:             ldc1 $f2, 0(t1) #f2 tiene el elemento de B
51:
52:             madd.d $f6, $f6, $f2, $f0 #f6 tiene A[i, j]*B[j, i] + A[i-1, j-1]*B[j-1,
53:
54:             addi t5, t5, 1
55:             j iter_n
56:         inc_columna:
57:         #Luego de iterar la fila de A y la columna de B, guardo en
58:         # C[ij] la suma acumulada.
59:         mul t8, t3, a3  #t8 = fila * n
60:         add t8, t8, t4  #t8 es fila*n + columna es indice de C
61:         sll t2, t8, 3   #t2 = t8*8
62:         add t2, a2, t2  #t2 tiene la direccion de C[t8]
63:         sdc1 $f6, 0(t2)
64:         addi t4, t4, 1
65:         j iter_columnas
66:     inc_fila:
67:     addi t3, t3, 1
68:     j iter_filas
```

```
   69:
   70: salir:
   71:      lw fp, 4(sp)
   72:      addiu sp,sp,8
   73:      jr ra
   74:
   75:      .end mmult
```

D. Script de ejecución

```
 1: #!/bin/bash
 2:
 3: RUTA_FUNCION=$1
 4: CLEAR_CACHE=$2
 5: RUTA_INPUTS=$3
 6:
 7: FLAGS_CACHEGRIND_I=$5
 8: FLAGS_CACHEGRIND_D=$6
 9: mkdir -p obj
10: mkdir -p asm
11:
12:
13: if [[ "$RUTA_FUNCION" == *.c ]]
14: then
15:     gcc -g -S $RUTA_FUNCION -o asm/mmult.s;
16:     sed -i '/.file 1/,+1d' asm/mmult.s
17:     es_assembler="No"
18: else #assembler
19:     cp $RUTA_FUNCION asm/mmult.S
20:     es_assembler="Si"
21: fi
22:
23: REAL_PATH=$(realpath asm/mmult.*)
24:
25: gcc -g -c asm/mmult.* -o obj/mmult.o
26: if [[ "$CLEAR_CACHE" == "clear_cache=yes" ]]
27: then
28:     sed -i '/#define CLEAR_CACHE 0/c\#define CLEAR_CACHE 1' src/benchmark.c
29: else
30:     sed -i '/#define CLEAR_CACHE 1/c\#define CLEAR_CACHE 0' src/benchmark.c
31: fi
32:
33: make
34:
35: RUTA_LOGS=$4
36: func_name=$(basename $RUTA_FUNCION | cut -d'.' -f1)
37: tipo_cache=$(echo ${FLAGS_CACHEGRIND_D:5} | tr , _)
38: clear_cache=$(echo $CLEAR_CACHE | tr = _)
39:
40: ruta_final=$RUTA_LOGS/$func_name/$tipo_cache/$clear_cache
41: mkdir -p $ruta_final/raw
42:
43: ARCHIVO_LOG_RESULTADOS=$ruta_final/log.txt
44: GREEN='\033[0;32m'
45: NC='\033[0m' # No Color
46:
47: for filename in `ls $RUTA_INPUTS/*.txt | sort -V`; do
48:     /opt/valgrind/bin/valgrind --tool=cachegrind --log-file=/dev/null $FLAGS_CACHEGR
49:
50:     dimension=$(echo $filename| tr -cd '[[:digit:]]')
51:     echo -n $dimension" " >> $ARCHIVO_LOG_RESULTADOS
52:     echo -n $filename" " >> $ARCHIVO_LOG_RESULTADOS
53:     echo -n $CLEAR_CACHE" " >> $ARCHIVO_LOG_RESULTADOS
54:     echo -n $RUTA_FUNCION" " >> $ARCHIVO_LOG_RESULTADOS
55:     echo -n $FLAGS_CACHEGRIND_I $FLAGS_CACHEGRIND_D" " >> $ARCHIVO_LOG_RESULTADOS
56:
57:     archivo_log_raw=$ruta_final/$dimension.txt
58:     /opt/valgrind/bin/cg_annotate cachegrind.out.* $REAL_PATH > $archivo_log_raw
59:     tail -n2 $archivo_log_raw | head -n1 >> $ARCHIVO_LOG_RESULTADOS
60:     mv $archivo_log_raw $ruta_final/raw/$dimension.txt
61:     rm cachegrind.out.*
62:     printf "Dimension: $dimension, $func_name, $tipo_cache, $clear_cache ${GREEN}Fin
63: done
64:
65: if [[ "$es_assembler" == "Si" ]]
66: then
67:     mv asm/mmult.S $ruta_final/mmult.S
68: else
```

```
    mv asm/mmult.s $ruta_final/mmult.s
fi

rm obj/mmult.o
```

E. Enunciado

El enunciado correspondiente al presente trabajo práctico puede encontrarse en este apéndice.

Universidad de Buenos Aires - FIUBA
66.20 Organización de Computadoras
Trabajo práctico 2: Introducción a caches
2^o cuatrimestre de 2020

`$Date: 2020/11/15 12:57:47 $`

1. Objetivos

Estudiar el comportamiento de los sistemas de memoria cache utilizando una serie de escenarios de análisis o *benchmarks* descriptos a continuación.

2. Alcance

Este trabajo práctico es de elaboración grupal, evaluación individual, y de carácter obligatorio para todos alumnos del curso.

3. Requisitos

El trabajo deberá ser entregado personalmente, en la fecha estipulada, con una carátula que contenga los datos completos de todos los integrantes, un informe impreso de acuerdo con lo que mencionaremos en la sección 6, y con una copia digital de los archivos fuente necesarios para compilar el trabajo.

4. Descripción

En este trabajo estudiaremos el comportamiento de una serie de configuraciones de sistemas de memoria cache, analizando la ejecución de un programa que permite multiplicar matrices cuadradas con cachegrind `cachegrind`.

Nombre	Tipo	Asociatividad	Parámetros `cachegrind`
C1	DM	Trivial	`--I1=32768,4,32 --D1=32768,1,32`
C2	2WSA	*2-WSA*	`--I1=32768,4,32 --D1=32768,2,32`
C3	4WSA	*4-WSA*	`--I1=32768,4,32 --D1=32768,4,32`

Cuadro 1: configuraciones para el sistema de memoria.

A lo largo de este TP, adoptaremos las 3 configuraciones para el nivel 1 y 2 del sistema de memoria cache indicadas en el cuadro 1.

En todos los casos la capacidad total será de 32 kbytes, y el tamaño de línea 32 bytes. Para cada una de estas configuraciones, deberá estudiarse el comportamiento de las mismas al ejecutar una implementación del algoritmo naïve de multiplicación de matrices cuadradas.

Para ello, deberá usarse el entorno QEMU del trabajo anterior [2], y la el programa cachegrind [3], una herramienta de *profiling* y simulación de sistemas de memoria cache multinivel que forma parte de la suite de software Valgrind [4].

4.1. Instalación de `cachegrind`

Debido a fallas en la versión de `cachegrind` suministrada dentro de la distribución de Linux que usamos en el TP, en este trabajo será necesario instalar una versión más reciente de esta herramienta en form manual. Para ello basta ejecutar el comando en la consola MIPS32:

```
$ gzip -dc valgrind-mips32-debian-stretch.tar.gz | (cd /opt/; tar -xvf -)
```

4.2. Funcionamiento de `cachegrind`

Para validar que la herramienta está funcionado, podemos compilar y ejecutar el ejemplo suministrado en `/opt/valgrind/share/fiuba/01-holamundo.S`:

```
$ cc -Wall -g -o /tmp/01-holamundo /opt/valgrind/share/fiuba/01-holamundo.S
$ /opt/valgrind/bin/valgrind --tool=cachegrind /tmp/01-holamundo
...
Hola mundo.
...
```

(Notar que en el ejemplo de arriba sólo hemos mostrado la salida propia del programa, y hemos suprimido las líneas generadas por el propio `cachegrind`). Este último comando ejecuta el binario `01-holamundo` dentro de la herramienta de profiling del sistema de memoria, y toma nota de la actividad realizada por el cache en el archivo `cachegrind.out.$pid`, donde `$pid` representa el número de proceso UNIX que tenía el proceso en el momento de realizar la simulación (en este caso `$pid` vale `3470`).

Para acceder a la información de *profiling* del sistema de memoria, basta con ejecutar el programa `cg_annotate`, indicando la ubicación del archivo con el código fuente del programa, y de esa manera poder acceder a las anotaciones línea por línea de la actividad del sistema de cache en nuestros programas MIPS32:

```
$ /opt/valgrind/bin/cg_annotate cachegrind.out.3470 \
           /opt/valgrind/share/fiuba/01-holamundo.S
...
--------------------------------------------------------------------------------
-- User-annotated source: /opt/valgrind/share/fiuba/01-holamundo.S
--------------------------------------------------------------------------------
Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw

-- line 4 ----------------------------------------
 .    .    .  .    .    .  .    .    .
 .    .    .  .    .    .  .    .    .   .text
 .    .    .  .    .    .  .    .    .   .align 2
 .    .    .  .    .    .  .    .    .
```

```
.    .    .  .    .    .  .    .    .   .globl main
.    .    .  .    .    .  .    .    .   .ent main
.    .    .  .    .    .  .    .    .  main:
.    .    .  .    .    .  .    .    .   .set noreorder
3    1    1  0    0    0  0    0    0   .cpload t9
.    .    .  .    .    .  .    .    .   .set nomacro
1    0    0  0    0    0  0    0    0   addiu sp, sp, -24
1    1    1  0    0    0  1    0    0   sw fp, 20(sp)
1    0    0  0    0    0  0    0    0   move fp, sp
1    0    0  0    0    0  1    0    0   .cprestore 0
.    .    .  .    .    .  .    .    .
1    0    0  0    0    0  0    0    0   li a0, 1
2    0    0  1    0    0  0    0    0   la a1, msg
1    0    0  0    0    0  0    0    0   li a2, 12
1    0    0  0    0    0  0    0    0   li v0, SYS_write
1    1    1  0    0    0  0    0    0   syscall
1    0    0  0    0    0  0    0    0   nop
.    .    .  .    .    .  .    .    .
1    0    0  0    0    0  0    0    0   move sp, fp
1    0    0  1    0    0  0    0    0   lw fp, 20(sp)
1    0    0  0    0    0  0    0    0   addiu sp, sp, 24
.    .    .  .    .    .  .    .    .
1    0    0  0    0    0  0    0    0   move v0, zero
1    0    0  0    0    0  0    0    0   jr ra
.    .    .  .    .    .  .    .    .   .end main
.    .    .  .    .    .  .    .    .
.    .    .  .    .    .  .    .    .   .rdata
.    .    .  .    .    .  .    .    .  msg:
.    .    .  .    .    .  .    .    .   .asciiz "Hola mundo.\n"

--------------------------------------------------------------------------------
Ir I1mr ILmr Dr D1mr DLmr Dw D1mw DLmw
--------------------------------------------------------------------------------
19    3    3  2    0    0  2    0    0  events annotated
```

5. Desarrollo

Usando las herramientas, deberá ejecutarse el algoritmo naïve de multiplicación de matrices repetidas veces, con tamaño de matriz progresivamente creciente: `n = 0, 2, 4, 8, ...`, en donde `n` representa la cantidad de filas y columnas de la matriz.

Deberán estudiarse los accesos que el algoritmo realiza sobre el sistema de memoria para acceder a los elementos de la matriz y calcular los productos internos de los vectores fila y columna.

Los resultados deberán representarse en un diagrama como el de la figura 1 que permita representar en la cantidad de desaciertos del cache L1D en función de `n`, como así también la tasa misses asociada.

Asimismo, cada grupo deberá realizar una predicción teórica del comportamiento del cache, el cual deberá contrastarse con las mediciones de cachegrind y documentadas en el informe (coincidan o no).

En este trabajo práctico no se requiere estudiar el comportamiento de otros niveles de cache (sólo L1D).

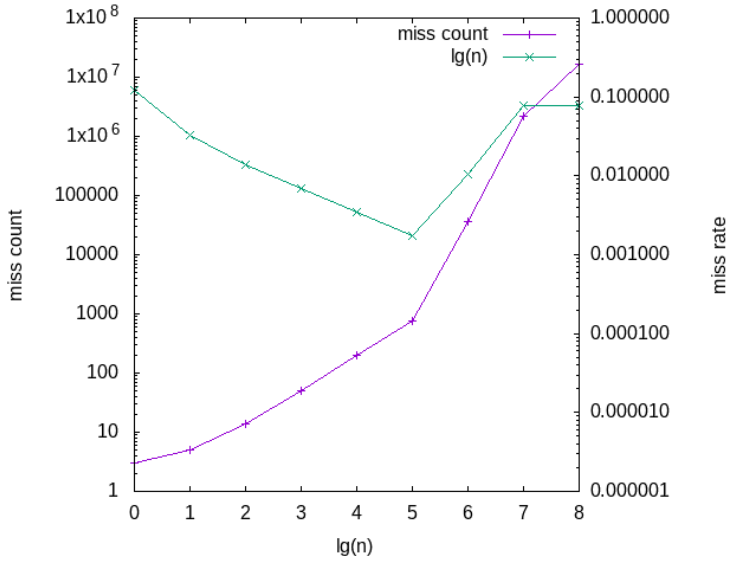


Figura 1: representación de cantidad y tasa de desaciertos para un dada combinación de algoritmo y configuración de cache.

En el informe del trabajo práctico, para caso de análisis deberá incluirse:

a) Detalle de todos los comandos usados para recolectar los datos de cada una de las corridas: invocación de `valgrind`, `cg_annotate`, etc.

b) Incorporar las anotaciones línea por línea de `cg_annotate` correspondiente al archivo `.S` del *benchmark* analizado.

c) Explicar en detalle porqué se produce la cantidad de desaciertos indicada sobre L1D para cada una de las configuraciones del sistema de memoria del cuadro 1[1].

d) Calcular la cantidad total de accesos a memoria, aciertos y desaciertos realizada por el cache L1D.

e) Calcular las tasa de desacierto para L1D asociada a cada combinación de *benchmark* y configuración del sistema de memoria.

f) Contrastar detalladamente los resultados de los cálculos con las simulaciones. **Justificar todo**.

En todos los casos, La ejecución deberá realizarse en el entorno MIPS32 simulado por QEMU. Asimismo, como en el TP anterior deberá usarse el modo 1 del sistema operativo para manejo de acceso no alineado a memoria [6].

6. Informe

El informe deberá incluir:

[1] Optativamente, se sugiere explorar también otras asociatividades, tamaños de bloque y variantes del algoritmo y contexo de ejecución

- Análisis detallado de cada uno de los *benchmarks*, siguiendo los lineamientos descriptos en las sección 5.
- El código fuente de todos los programas analizados en el TP.
- Instrucciones de compilacióm y ejecución de cada caso.
- Este enunciado.

7. Entrega de TPs

La entrega de este trabajo deberá realizarse únicamente a través del campus virtual de la materia [7] dentro del plazo de tiempo establecido.

Asimismo, en todos los casos, estas presentaciones deberán ser realizadas durante los días martes. El *feedback* estará disponible de un martes hacia el otro, como ocurre durante la modalidad presencial de cursada.

Por otro lado, la última fecha de entrega y presentación para esta trabajo es el martes 1/12.

Referencias

[1] Implementación de referencia para el algoritmo naïve de multiplicación de matrices. `https://drive.google.com/file/d/1gEVZ8d7IVMu9R3QUBlwmjQTXjXYEeJVJ/view?usp=sharing`

[2] Enunciado del Trabajo Práctico 1, segundo cuatrimestre de 2020. `https://drive.google.com/file/d/1dtWzlAwxxpLqwifaCwJ3_0WyYwvoavUd/view?usp=sharing`.

[3] Cachegrind: a cache profiler. `http://valgrind.org/docs/manual/cg-manual.html`.

[4] Valgrind: programming tool for memory debugging, memory leak detection, and profiling. `https://valgrind.org/`.

[5] Binarios de Valgrind para correr en QEMU MIPS32. `https://drive.google.com/file/d/1n4_b5xHjaA8dAmEiileoAOlZjjsunRWa/view?usp=sharing`.

[6] Controlling the kernel unalignment handling via debugfs (Linux/MIPS wiki). https://www.linux-mips.org/wiki/Alignment.

[7] Aula Virtual - Organización de Computadoras 86.37/66.20 - Curso 1 - Turno Martes. `https://campus.fi.uba.ar/course/view.php?id=649`